COMSC-210 Lab 8
Big Oh and Algorithm Efficiency

GOOD PROGRAMMING PRACTICES *show / hide*

ABOUT THIS ASSIGNMENT
In this assignment you will use big oh to predict the speed performance of program operations, and learn how to confirm your predictions with timing studies. Just a warning -- the timing tests shown here may not work with some versions of Linux!

After you complete this lab assignment, post the required files to the COMSC server so that you receive the proper credit for this 50-point lab. Your posted files are due at midnight of the evening of the due date indicated in the course outline. Use the "Submit your FA2015 work" link on the class website to post your file for this lab, using the **lab8** folder provided.

LAB 8a: Perform A Simple Timing Study `[ FileInput.cpp ]`
Purpose. In this lab you will apply the "4-cycle timing code" presented in the lecture topic 8 notes. The purpose is to prepare you for making big oh determinations and confirming them with timing studies.

Requirements. Write **`FileInput.cpp`**, applying the "4-cycle timing code" from the lecture notes, to the reading of the dvc-schedule.txt file from lab 5. Do not read the whole file -- just read `n` lines. Parse as you did in previous labs that used this same input file, but do not store the results in a data structure. Read lines, parse, and discard the results, timing how long it takes to do so for n lines. Start with `n = 5000` for the first cycle, and confirm `O(n)` -- that is, the time it takes to read the file is directly proportional to the number of lines read from the file.

Note that your computer may cache your input file, throwing off the timing for the first cycle. So run your program more than once to see it work right.

HINT: Create and recreate the `ifstream` object for EACH of the 4 cycles, OR refer to Burns' Comsc-110 textbook, chapter 10, for how to close and reopen a text file.

Submit the CPP to the class website for credit. Do NOT submit the TXT input file.

Program I/O. No input. Console output reporting timing results per the provided code.

Example.

```
1436 (expected O(n)) for n=5000
2742 (expected 2872) for n=10000
5442 (expected 5744) for n=20000
10828 (expected 11488) for n=40000
```

LAB 8b: Perform A Timing Study On Nested For-Loop Sorting `[ NestedForLoop.cpp ]`
Purpose. In this lab you will make your own determination for the big oh of nested for-loop sorting of an array, and confirm your determination.

Requirements. Write **`NestedForLoop.cpp`**, applying the "4-cycle timing code" from the lecture notes, to the sorting of a dynamic array of `doubles`. You decide what you expect the big oh to be, and what `n` to use for the first cycle. In each cycle, create the array and fill it with random values (using `srand` and `rand`). Then start the timer, perform the sort, and stop the timer. *Write code to verify that each number in the array is greater or equal to the number before it, starting with the 2nd number. Use `assert`, like this:*

```
for (int i = 1; i < n; i++) assert (a[i - 1] <= a[i]);
```

This way, there's not a lot of output to look through in order to verify that the sorting performed correctly.

Submit the CPP file to the class website for credit. But in the version you submit, don't make `n` so large that it takes more than a few seconds to run during grading!

Program I/O. No input. Console output reporting timing results per the provided code.

LAB 8c: Perform A Timing Study On The STL Array Sort Function `[ Sort.cpp ]`
Purpose. In this lab you compare the big oh performance of the sorting function provided in the STL to the nested for-loop.

Requirements. Rewrite `NestedForLoop.cpp` from the previous labs as **`Sort.cpp`**, applying the "4-cycle timing code" from the lecture notes, to the sorting of a dynamic array of `doubles`. You'll have to decide the big oh based on research and/or a process of elimination. Decide what `n` to use for the first cycle -- it does not have to be the same as the previous lab, because you can expect the STL to be fast. As before, in each cycle, create the array and fill it with random values (using `srand` and `rand` -- remember to call rand once after srand to "prime" the sequence.). Then start the timer, perform the sort, and stop the timer. Report the results and verify that the sort worked.

The sort function is in the C++ `algorithm` library. The function call for an array named `array` is this: `sort(array, array + n);` -- the same `n` used to declare the array.

Submit the CPP file to the class website for credit. But in the version you submit, don't make `n` so large that it takes more than a few seconds to run during grading!

Program I/O. No input. Console output reporting timing results per the provided code.

How to pause a console program: *show / hide*

GRADING RUBRIC *show/hide*
